DOI 10.12694/scpe.v14i4.929
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 215–233. http://www.scpe.org

MOBILES FOR SENSING CLOUDS: THE SAAAS4MOBILE EXPERIENCE *

SALVATORE DISTEFANO† GIOVANNI MERLINO ‡ AND ANTONIO PULIAFITO§

Abstract. Smart devices, and mobiles in particular, are at the forefront of several hot new trends in ICT, such as the Internet of Things and service computing. Cloud computing is another paradigm generating a great deal of offshoots, some of which are aimed at enabling novel services and applications by exploiting its ubiquity and flexibility in combination with sensors and the (meta)data they produce about phenomena, events and other interesting items about the physical world. In this context, the authors, propose a new way to orchestrate devices, in particular SNs and mobiles, as resources to build up Clouds of sensors, reverting the current wisdom about mobile Clouds, i.e. the integration of feature-rich devices into the Cloud fabric as mere clients to one where personal / wearable devices are actively involved into a "sensing" Cloud, forming a fully feedback-enabled ecosystem. The main aim of the Sensing and Actuation as a Service (SAaaS) approach is therefore to implement such a Cloud by enrolling and aggregating sensing resources from sensor networks and personal, mobile devices. A device-centric approach is embraced as in IaaS Clouds: once collected, the physical (sensing) resources are abstracted and virtualised and then provided elastically, on-demand, as a service to end users, including facilities for customization of the (hosting) embedded platform.

A key point of the SAaaS approach is the abstraction of resources, i.e. providing a uniform way to access to and interact with the underlying physical nodes. In this paper we focus on the low-level interaction with sensing resources in SAaaS, restricting the scope to mobiles, thus providing details on theoretical and design aspects as well as technical and implementation ones. In particular, we report on an implementation of the SAaaS low-level modules on Android devices, the SAaaS4Mobile one, providing architectural descriptions of the main modules, implementation guidelines and discussing through a preliminary implementation evaluation the effectiveness of the approach.

Key words: Cloud, sensors and actuators, sensing abstraction and virtualisation, OGC Sensor Web Enablement, Android.

1. Introduction and Motivations. Since their introduction and adoption, in early '90s, mobiles strongly impacted on everyday life changing, sometimes improving, the lifestyle. This transformation process is based on the advancements of network and processor technologies, following new trends and forms of interactions. Current smartphones have computing, storage and sensing capacities that can be compared to laptop or desktop, as a new, radical, interpretation of personal computing, the personal device or PDA frontier. This state of affairs has unlocked new ideas and paved the way for rethinking and reinterpreting foundational technologies such as Internet, driving efforts towards the Internet of Things (IoT), or service engineering at the basis of the Cloud computing and the Web 2.0.

At a lower scale, this also inspired us in developing an idea mixing aspects of both IoT and Cloud, involving any form of sensing resource such as sensor networks as well as standalone smart devices into a wide-area (geographic) sensing infrastructure. Our idea was to gather and collect sensing and actuation resources from contributing nodes, either sensor networks or personal, mobile devices, following a volunteer-based approach to build up a scalable sensing infrastructure, on-demand providing sensing resources to end users according to their requirements, as computing resources in Infrastructure as a Service (IaaS) Clouds, towards sensing Clouds. Thus, in previous work [9, 10] we proposed the Sensing and Actuation as a Service (SAaaS) framework to deal with the issues arising in sensing Clouds, mainly referring to resource abstraction and virtualisation, enrolment, indexing, discovery and management.

According to the SAaaS approach, the sensing Cloud provider that builds up and manages the infrastructure, has to provide actual sensing and actuation resources, even if abstracted and virtualised through a specific framework, to the user. Users may therefore handle, manage and customize (virtual) sensing resources at their will, according to their needs, for example for inclusion into an existing sensor network they administrate, e.g. when not able anymore to guarantee coverage of a certain area. This way the user resorts to a sensing Cloud

*This work is partially funded by PhD programme under grant PON R&C 2007/2013 "Smart Cities" and by Simone project under grant POR FESR Sicilia 2007/2013 n. 179. The authors would also like to acknowledge networking support by the COST Actions IC1203 - ENERGIC and IC1303 - AAPELE.

†Dip. di Elettronica, Informazione e Bioingegneria Politecnico di Milano Piazza L. Da Vinci 32, 20133 Milano, Italy. Email: salvatore.distefano@polimi.it

‡Dip. di Ingegneria, Università di Messina, Contrada di Dio, 98166 Messina, Italy. Email: gmerlino@unime.it

Dip. di Ingegneria Elettrica, Elettronica e Informatica, Università di Catania, Viale Andrea Doria 6, 98166 Catania, Italy. Email: giovanni.merlino@dieei.unict.it

§ Dip. di Ingegneria, Università di Messina, Contrada di Dio, 98166 Messina, Italy. Email: apuliafito@unime.it

once the resources are not enough, asking for further sensing resources, in a Cloudbursting fashion. We defined our approach as *device-centric* against the *data-centric* one, which aims at providing the user just with, more or less elaborated, sensed data, hosted by traditional Cloud providers [12], thus a Software as a Service (SaaS) approach at the core. A really key difference between the two approaches is that in the device-centric one the user obtains actual sensing resources that could be configured and managed according to user requirements, while in the data-centric one the user can just retrieve data gathered and offered by the provider itself, since the sensing resource may not be handled directly by the former, because it is managed by the provider itself.

One of the key features of a device-centric approach is the abstraction of sensing resources, i.e. techniques for abstracting details and mechanisms from heterogeneous hardware solutions, to access, interact and communicate with the sensing resources. In the SAaaS paradigm such functionalities are delegated to the Hypervisor component, which also provides virtualisation mechanisms. This kind of low-level albeit mediated (shared-)access to resources is what bears most of the challenges to be investigated to design and implement an IaaS-like Cloud such as the SAaaS one.

In this paper we specifically focus on the low-level interaction with sensing resources belonging to SAaaS-enabled infrastructure, as a first step for a bottom-up implementation of the framework. Specifically, we start our investigation by considering smart devices only, thus developing a first prototype of the SAaaS abstraction layers for mobiles, the *SAaaS4Mobile*.

When mobiles are involved, certain advantages of our device-centric approach are more easily recognizable, e.g. opening up opportunities for direct involvement of (i.e. notifications to) the device owner / resource contributor, a chief requirement for mobile crowdsensing scenarios. On the other hand, for all its worth, enrolment of mobiles bears many unique challenges, to be tackled by means of volunteering approaches, in order to satisfy certain predefined (e.g. SLA-mandated) reliability and QoS requirements [8, 11]. Moreover, reward systems are one of the approaches to be called for mobiles, as very peculiar motivations play their role when dealing with volunteering owners of resources with such hard constraints (e.g. relatively fast battery depletion [4]).

We therefore characterize the overall Hypervisor architecture [10] for mobiles, since some of the components previously identified as distinct modules in the general case, i.e. also including SNs, merge when dealing with standalone smart devices. Then, we focus on the main, mandatory components of the SAaaS4Mobile Hypervisor architecture, investigating and implementing a solution with specific regards to the abstraction layers modules. The implementation of SAaaS4Mobile abstraction modules has been tackled for Android mobiles only, at this stage, starting from well-known standards as the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) [23] ones. To demonstrate the feasibility of the SAaaS approach the SAaaS4Mobile core modules implementation has been tested through a proof of concept in which the performance of the main low level operations have been measured. The values thus obtained fosters further developments of the paradigm, since they provide useful insight on its effectiveness.

In order to explain in detail our work we organized the paper as follows. Section 2 provides an overview of the state of the art on related work, mainly focusing on some specific aspects provided as background. The Hypervisor architecture is discussed in cf. Sect. 3, with particular emphasis on the standalone device characterization into SAaaS4Mobile. The corresponding architecture for abstraction layer modules is then discussed in cf. Sect. 4, also describing possible interactions with the system from a dynamic-behavioural perspective in cf. Sect. 5. Details on the implementation of a preliminary version of the SAaaS4Mobile stack on Android and SWE standard environments are reported in cf. Sect. 6 with the evaluation of a proof of concepts. Finally, cf. Sect. 7 summarizes the paper providing hints for future work.

2. Preliminary Concepts.

2.1. Related Work. In sensing environments, software abstraction layers allow to address interoperability and communication issues [21, 27] to enable the dynamic reconfiguration of sensor nodes [19, 2], and to manipulate sensor data [1, 20].

Some solutions are specifically conceived for building up networks of mobiles' sensors. The Mobile phone Sensor Network [6] allows to collect observations from Bluetooth-enabled sensors on mobiles and send them to a database through an OGC SWE SOS. Similarly, [18] allows to perform the injected measurements and to express them in SWE-compliant format, also resorting on the mobile computing and storage resources.

SWE [23] is a standards' suite for achieving abstraction and interoperability in sensor networks, widely used [6, 18, 5] in the implementation of sensing Web services as well as to address abstraction issues in *virtual sensor networks* [24]. It comprises several standards such as the *Sensor Model Language* for the description of sensor systems and processes associated with sensor observations; the *Observations & Measurements* to express observations and measurements through standard Web service interfaces; the *Sensor Observations Service* (SOS) to collect observations and system information; the *Sensor Planning Service* (SPS) to plan observations; the *Sensor Alert Service* (SAS) to publish and subscribe sensor alerts; the *Web Notification Service* (WNS) to asynchronously deliver messages and alerts from SAS and SPS.

In [27], a framework to enable management of physical sensors on IT infrastructure, abstracting and virtualising them into virtual sensors is provided. In [25] an infrastructure for connecting sensor networks and applications is proposed, allowing to select physical sensors in wide-geographic sensor networks that can be implemented as suggested in [13].

A Semantic Web Architecture for Sensor Networks (SWASN) is proposed in [17], specifically dealing with inference on the sensor data collected by several heterogeneous SNs. In [5] a service-oriented framework to integrate heterogeneous sensors and virtual ones is described, starting from the SWE SPS, WNS, and SOS.

In [3] Android mobile sensors are categorized into two different classes depending on the functionality provided (common an complimentary) and are considered as potential providers of raw data. To recognize user context information the mobile phone infrastructure of [14] can be used, e.g. for monitoring physical actions performed by users such as walking and running. Similarly, [26] allows to recognize physical activities of mobile phone owners-users.

2.2. Background. In this section we explore the topics about the background of SAaaS4Mobile: a high-level description for the SAaaS paradigm, including a depiction of its layering, followed by a subsection on standards relevant to our effort, in this case SWE by OGC, and by details on the specific platform to be addressed, i.e. Android.

2.2.1. SAaaS. The main aim of SAaaS [9] is to adapt the IaaS paradigm to sensing platforms, bringing to a Cloud of sensors, where sensing and actuation resources may be discovered, aggregated, and provided as a service, according to a Cloud provisioning model.

The inclusion of sensors and actuators in geographic networks as Cloud-provisioned resources brings new opportunities with regards to contextualization and geo-awareness. By also considering mobiles, possibly joining and leaving at any time, the result can be a highly dynamic environment. The issue of node churn can only be addressed through volunteer contribution paradigms [9, 7]. Furthermore, the SAaaS has to manage contributions coming from sensor networks, mobiles or any other "smart" device equipped with sensors and actuators, to ensure interoperability in a Cloud environment. It must also be able to provide the mechanisms necessary for self-management, configuration and adaptation of nodes, without forgetting to provide the functions and interfaces for the activation and management of voluntarily shared resources.

The SAaaS reference architecture [9] comprises three modules, Hypervisor, Autonomic Enforcer and VolunteerCloud Manager, shown in cf. Fig. 2.1. The *Hypervisor* allows to manage, abstract, virtualise and customise sensing and actuation resources that could be provided by enrolling either mobile device or SN *nodes*. Among key features are: abstraction of devices and capabilities, virtualization of abstracted resources, communications and networking, customization, isolation, semantic labeling, and thing-enabled services. All these features are presented in the next section. At a higher level with respect to the Hypervisor, the *Autonomic Enforcer* and the *VolunteerCloud Manager* deal with issues related to the interaction among nodes. The former is responsible of the enforcement of local and global Cloud policies, subscription management, cooperation on overlay instantiation. The VolunteerCloud Manager is in charge of exposing the Cloud of sensors via Web service interfaces, indexing of resources, monitoring Quality of Service (QoS) metrics and adherence to Service Level Agreements (SLAs).

2.2.2. OGC: Sensor Web Enablement. The OGC provides a large number of specifications, among which we can find the Sensor Web Enablement (SWE) family of standards. Designed for the management of sensor data on the web, as mentioned in [23], a unique framework of open standards for exploiting Web-connected sensors and sensor systems of all types is the focus of the specifications.

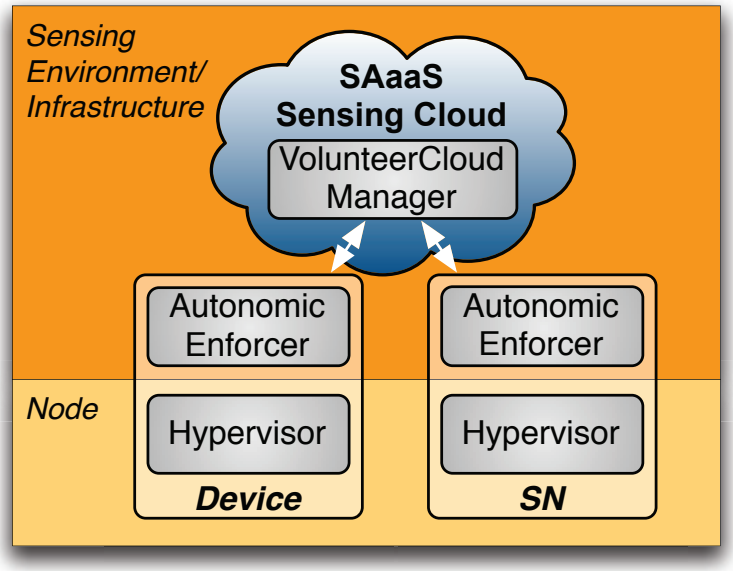


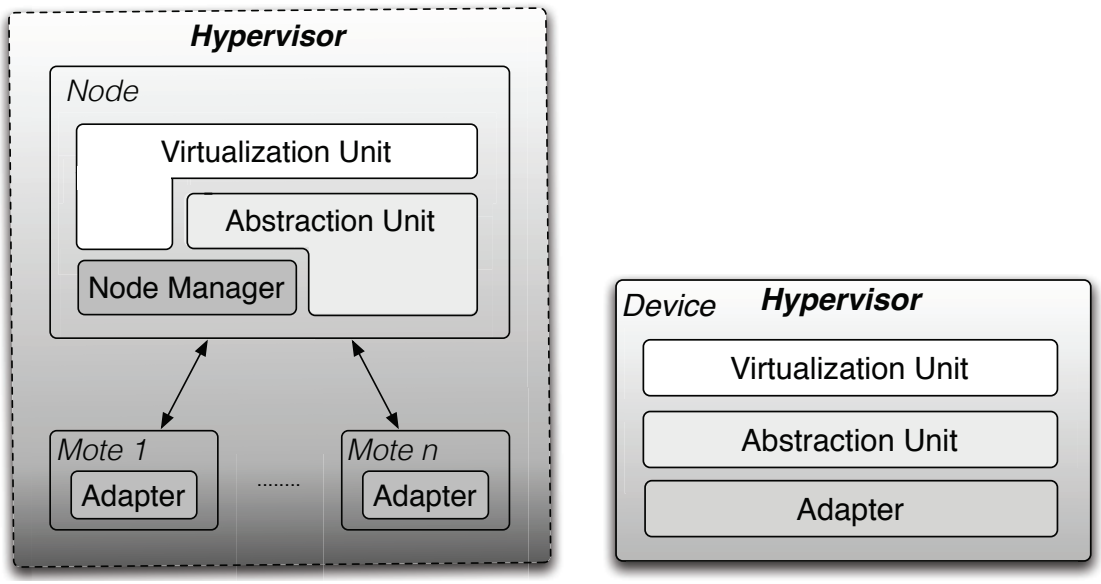
FIG. 2.1. *SAaaS reference architecture*

SWE standards aim at making all types of Web sensors, instruments, probes, and imaging devices accessible and controllable on the Web. The SWE framework is composed of seven standards, four of them have been approved as official standards by the OGC members.

- SensorML: it is a language based on XML schema to describe the sensor systems. It encodes a lot of features for sensors, such as discovery, geolocation processing observations, mechanisms for sensor programming, subscriptions to sensor alerts. In particular, it provides standard models and XML schemas to describe processes, and instructions for obtaining information from observations. SensorML enables discovery, access and query execution for the processes and sensors it models.
- Observations & Measurements (O & M): this model in particular is featured in the SOS specification, coupled with an XML encoding for observations and measurements originating from sensors, and archived in real-time. It provides standardized methods for accessing and exchanging observations, alleviating the need to support a wide range of sensor-specific and community specific data formats.
- Sensor Observation Service (SOS): it corresponds to the Observation Agent specified in the previous section. This is the service responsible of the transmission of measured observations, from sensors to a client, in a standard way that is consistent for all sensor systems including remote, in-situ, fixed and mobile sensors. It allows the customer to control the measurement retrieval process. This is the intermediary between a client and an observation or near real-time sensor repository.
- Sensor Planning Service (SPS): it corresponds to the Planning Agent, whose design and implementation are also addressed in this paper.

2.2.3. Mobile OS and IDE. Android is a mobile operating system developed by Google, in reality a software platform composed of five parts [16, 28].

- Applications: the platform provides a set of core applications, which includes email client, SMS app, contacts app, calendar, mapping, Web browser, etc.
- Application Framework: the base framework for developing Applications in Android. It is composed of a set of tools enabling the realization of applications. Moreover, security implications and privacy protection, among core concerns about such a device-driven approach, are mostly taken care by the Application Framework itself, deviating most of the attention the topic, here unaddressed accordingly, would otherwise deserve, e.g. in mixed / WSN-powered topologies.
- C/C++ Library: Android includes a set of C/C++ libraries, used by various components of the Android



(a) SN (b) Device

FIG. 3.1. *The Hypervisor modular architecture.*

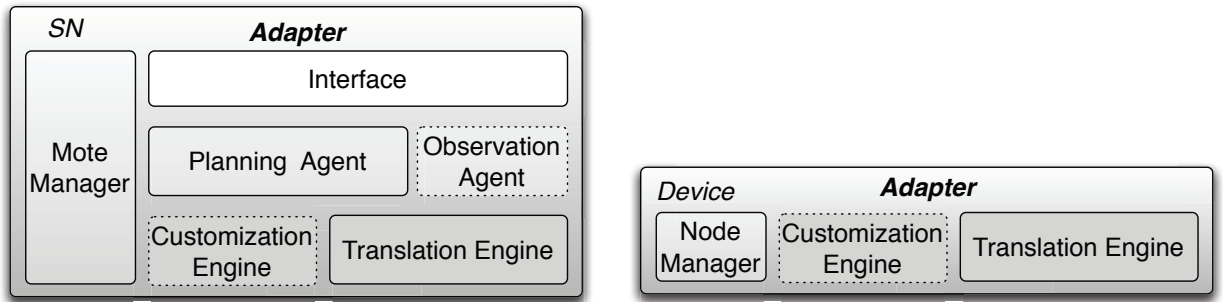
system. These capabilities are exposed through the Application framework.

- Android Runtime Library: It provides a big part of functionalities available in the core libraries of the Java programming language.
- The Linux kernel: Android relies on a Linux 2.6-based kernel for core system services such as memory management, process management, network stack, security, driver model, etc. The core also acts as a abstraction layer between the applications and the hardware.

In order to develop an application, Android offers a Software Development Kit (SDK) and a Native Development Kit (NDK) [15]. The SDK provides a large number of development tools, needed to build, test, and debug Android applications, including an emulator, able to emulate an Android mobile.

On the other hand, the NDK is the Android operating system API to natively develop in the language of the target architecture, as opposed to the Android SDK, which provides (Java) bytecode-based abstractions, thus hardware independent ones. The NDK helps developers integrate (typically, ARM) native code in their applications to exploit the performance offered by accelerated operations in the processor. Based on the Dalvik virtual machine, developers can embed C or C++ code to reuse some classes that have already been developed for other systems.

3. The Hypervisor. A *Hypervisor* [10] can be viewed as the foundational component of our device-driven approach to infrastructure-focused Clouds of sensors: it manages the resources related to sensing and actuation, introducing layers of abstraction and mechanisms for virtualization. It works at the node level, which could be either a whole sensor network, under a unique administrative domain, or a set of sensors, as built-in to a standalone device. In other words, also referring to cf. Fig. 2.1, a *node* could be either a whole sensor network, composed of several sensor boards or *motes* managed by a sink and/or a gateway, or a personal device that could be more or less smart and thus can be equipped with one or more sensors. The Hypervisor functionalities should fill this gap, dealing with such heterogeneity, hiding it to the above modules of the SAaaS reference architecture. In the SN case, it is therefore necessary to split the Hypervisor architecture between the SN node and mote layers. This way, all the motes composing an SN should have installed a specific Hypervisor module locally managing the motes, coordinated by the high level modules of the Hypervisor deployed on the node/SN gateway. This kind of two-level separation of concerns and assignment of operations descends also from the



(a) SN (b) Device

FIG. 3.2. *Adapter modules.*

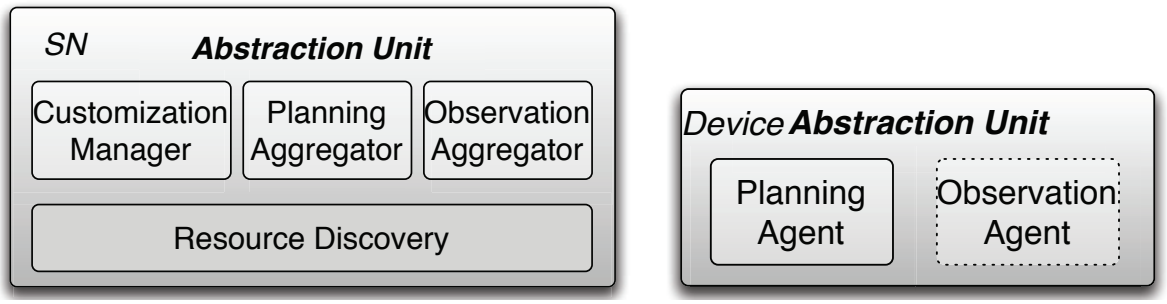
need for certain duties to be (self-)managed through autonomic approaches, typical of distributed entities.

A high-level, modular view of the Hypervisor architecture comprises four main building blocks, when dealing with SNs: Adapter, Node Manager, Abstraction Unit and Virtualization Unit as shown in cf. Fig. 3.1(a), collapsing down to three out of four components, when standalone devices (e.g. mobiles) are involved, as depicted in cf. Fig. 3.1(b).

At the bottom of the Hypervisor, there is the *Adapter*, which plays several distinct roles through its modules. The *Translation Engine* is a platform-specific driver, in charge of converting the high-level directives in native commands. The Hypervisor is also appointed for processing requests for reconfiguration of the device, using the (optional) *Customization Engine*, an interpreter able to execute on the sensing device the code needed to tailor the sensing activities to customer-mandated requirements. Yet, the most important duty this layer-spanning module has to cope with consists in providing mechanisms for the customer to establish an out-of-band (i.e. not Interface- or Agent-mediated) channel to the system, for direct interaction with either the resources (e.g. for Agent-agnostic collection of observations) or low-level modules (e.g. the Customization Engine), thus pinning it as a mandatory component of the architecture.

The *Node Manager* works only at the node level and is in charge of the basic sensing resource operations and mandating policies, in cooperation with the Adapter *Mote Manager* that replicates its functionalities at mote level. In standalone device the two modules are collapsed into the Adapter Node Manager (cf. Fig. 3.2(b)). It is important to remark that the depiction (cf. Fig. 3.1(a)) of the Virtualization Unit for SNs is L-shaped because it can work directly over the Agent-hosting Adapter in selected cases, e.g. when dealing with a degenerate SN made up of a single mote. In such cases, an autonomic approach is adopted delegating some management tasks of the Adapter to the Mote Manager running on the mote-side, performing specific operations such as power-driven self-optimization with the Node Manager. The Mote Manager is not needed on a standalone device, where the Node Manager itself makes up for the combination of the two modules.

The *Observation Agent*, featured in the Adapter for SNs (cf. Fig. 3.2(a)) and in the Abstraction Unit for mobiles (cf. Fig. 3.3(b)), is in charge of requesting, retrieving, and eventually pre-processing measurements. The *Planning Agent* (PA) pushes requests for actions (*tasks*) to the device. It is featured in the Adapter or the Abstraction Unit, depending on the kind of topology, as also for the Observation Agent. The requests leaving the PA are for preparing the resource to carry out a variety of duties (reservation of functionalities, tuning of parameters, scheduling of observations). These commands allow management of operating parameters such as duty cycle, sampling frequency, etc. Although providing useful and standardized mechanisms, the Observation Agent is not strictly needed to let customers exploit sensing infrastructure. Indeed the PA is enough to handle the physical (or virtualized) resources as long as a working bidirectional communication channel is established between the client and the mote or mobile hosting the sensing device. Such a facility would then be enough to let the customer do what is needed for getting and storing observations e.g. even build a client-side version of a SensorML-compliant module for management of observations, synchronously working over the channel if required.



(a) SN (b) Device

FIG. 3.3. *Abstraction unit modules.*

Furthermore, in SN deployments, the Adapter has to expose a standard-compliant customer-friendly *Interface* to on-board resources.

The Adapter provides its functionalities to the upper level *Abstraction Unit*. As can be seen comparing cf. Fig. 3.3(a) with cf. Fig. 3.2(a), with regards to SNs it replicates planning and observation facilities, modeled after those featured in the Adapter, but on a node-wide scale, combining the pool of resources of the whole SN. In particular, the *Observation Aggregator* exposes all resources from the nodes and the *Planning Aggregator* manages this set, sending combos, i.e. combination of commands, and tracking exit codes, eventually reacting to (partial) failures by triggering apt adjustments. The *Resource Discovery* module, which offers an interface to the motes, actively gathering descriptions of underlying resources and forwarding the results to the Aggregator modules. The *Customization Manager* acts as an orchestrator for customization engines located on the motes.

With regards to mobiles, the architecture of the Abstraction Unit degenerates from the one depicted in cf. Fig. 3.3(a) to the one in cf. Fig. 3.3(b), where only planning- and observation-oriented modules are part of the unit, now named Agents for taking on the same role as their counterparts in the Adapter for SNs (cf. Fig. 3.2(a)), thus leaving only the lowest layer of the latter in the version of the Adapter geared towards standalone mobiles (cf. Fig. 3.2(b)).

All components of the Abstraction Unit in cf. Fig. 3.3(a) are mandatory (solid line border), as those are needed to coordinate operations of the corresponding mote-side modules (when present), while the Observation Agent in cf. Fig. 3.3(b) is optional (dashed line border), as its counterpart is in cf. Fig. 3.2(a).

The highest level of the Hypervisor is the *Virtualization Unit*, name after the Virtual Machine Monitor to highlight its role as a manager of the lifecycle of virtualized resources instance. This includes APIs and functionalities for virtual instance creation, reaping and repurposing, as well as for boot- (defined statically) and run-time (dynamically) parameters discovery and tuning in accordance to contextualization requests.

4. Basic Abstraction Modules. Aim of this section is to provide details on the main modules implementing abstraction and adaptation facilities of the SAaaS4Mobile framework on mobiles.

4.1. Translation Engine and Node Manager. The SAaaS4Mobile Node Manager of cf. Fig. 3.2(b) is mandatory exclusively for its role as an out-of-band conduit, e.g. to the Customization Engine, where otherwise the only way to interact with sensing resources is mediated by the Planning Agent, leaving no path to mould the platform itself to the sensing requirements at hand. A way to establish this channel then, under a mobile platform with typical features such as Android, can be through platform-provided facilities, i.e. Google Cloud Messaging.

With regards to the Translation Engine, any sensing-related APIs provided by the node should be used when available since developing the Agents against those frees us from the need to implement a layer for command translation. For example, Android provides platform-specific Sensing APIs. Yet we cannot fully dispose of the Translation Engine, because not all onboard devices, which could be leveraged for sensing, are exposed as such (i.e. under the sensing APIs). This way the Translation Engine exploits the platform-standard (e.g. Android

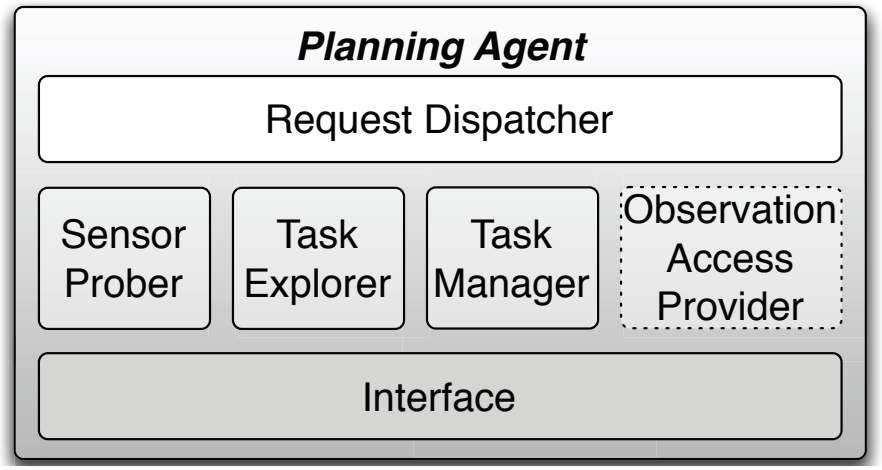


FIG. 4.1. *Planning Agent architecture*

and Linux) low-level tools and mechanisms in order to export these resources under an extended set of devices the sensing APIs know about, easing the job of the Sensor Prober, e.g. just letting it enumerate resources through the sensing APIs.

4.2. The Planning Agent. The SAaaS4Mobile Abstraction Unit, as shown in cf. Fig. 3.3(b), is mainly composed of the Planning Agent (PA) that is the only mandatory module of the unit, whereas the same building block resides in the Adapter when considering SNs. In the following, the details about the PA and its components are to be considered valid in both cases. The PA works side-by-side with the Observation Agent, complementing its features. Unlike the latter, engaged in providing upper layers with XML-encoded measurements (*observations*), sampled while driving the sensing resources, the former is mainly devoted to tune sampling parameters according to user-defined preferences, still to be interfaced with by means of extensible standards-compliant encoding of requests for *tasks*, and corresponding responses. Other than tuning, tasks for scheduling of observations can be consumed by the PA: it may be following a predefined schedule, or upon the occurrence of a particular event, or simply a request from a client. The main aim of this effort is exposing all underlying knobs to make them available for customers to operate on transparently.

In order to meet the aforementioned requirements, an architecture comprising the six modules shown in cf. Fig. 4.1 has been designed: a Request Dispatcher, a Sensors Prober, a Task Explorer, a Task Manager, an Observation Access Provider and an Interface. The *Request Dispatcher* has to identify and demultiplex a request to the modules underneath. The *Interface* has to interact with the SAaaS4Mobile Adapter services, i.e. the Customization Engine, the Translation Engine and the Node Manager.

The *Sensor Prober* is in charge of enumerating all the sensors and actuators within a sensors platform, however rich and complex, by low-level platform-specific system probing. These sensors are then identified according to their types, supported observation facilities and sampling specs, overall (nominal) features and manufacturing details (brand, model, etc.).

The *Task Explorer* is responsible for enumeration of available tasks, to be provided by probing sensors as listed by the aforementioned Prober. In terms of tasks, those related to parameter tuning for sensing resources logically differ depending on the sensor type and technology, it is therefore possible to e.g. plan retrieval of temperature samples from a thermometer, once a certain threshold has been exceeded, change the relative position and the focal length of a camera, or simply schedule reading of sensor observations at fixed intervals, etc. Moreover, in order to assess feasibility of a certain task, among the ones enumerated for selection, the sensor has to be queried and provide (runtime) confirmation, or else denial, of availability for servicing (or reservation thereof). It's then up to the querying party to decide what to do after feasibility assessment for the task under consideration.

The *Task Manager* controls tasks' lifecycle, since feasibility assessment through reservation/submission stages, then following up, and acting upon, running task progress. Due to the number of, and dependencies for,

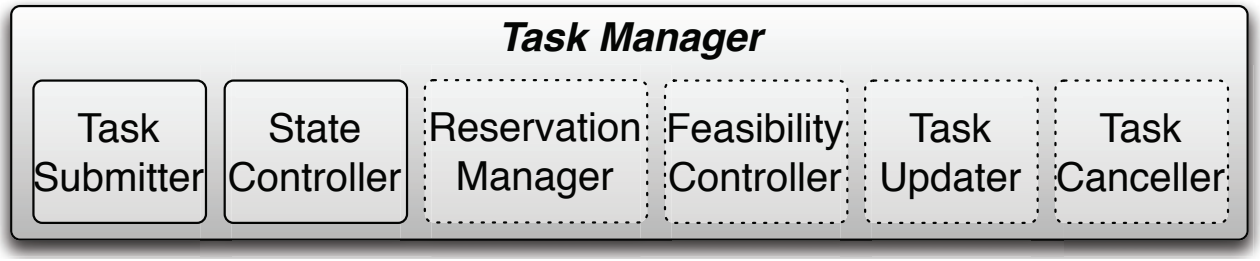


FIG. 4.2. *Task Manager architecture*

the operations involved, the Task Manager duties have been assigned to six modules as shown in the architecture of cf. Fig. 4.2. Two of them are mandatory, the rest are optional.

Task Submitter and *State Controller* implement mandatory functionalities. Their roles are, respectively, to enable users to set all (mandatory) parameters for a specific task before submission to a sensor, and submit it when ready, and to follow up the processing of the task, alerting any agent, subsequently querying about availability for task execution, about its (busy) status until completion. The optional modules are instead: Reservation Manager, Feasibility Controller, Task Updater and Task Canceller. These modules provide additional facilities for control on running (or yet to be scheduled) tasks to process.

If needed, a user may reserve a task for a period of time, during which he/she gets exclusive access to the underlying resource, as no other user can submit or reserve it. The task will then be executed as soon as the user confirms for the real processing stages to commence. The *Reservation Manager* is responsible for both reservation of tasks, and its confirmation. The *Feasibility Controller* has to check if a task is feasible, as detailed above. The feasibility of a task depends on the availability of any resource essential for task servicing, e.g. if not still allocated due to a previous request.

Then, the *Task Updater* is in charge of updating configuration parameters of a task, if some modifications have to be pushed after tasks enter into processing stages. Lastly, the *Task Canceller* empowers users to stop and therefore retire a task, when already submitted or under reservation.

Finally, once a task has been serviced, the resulting observation gets stored. Any observation will be accessible through the Observation Agent only. In terms of observations, the sole duty up to the PA lies in the *Observation Access Provider* ability to provide endpoints to access measurements. Being dependent on the Observation Agent, it is an optional component, required only if the latter is implemented in the Abstraction Unit.

5. A Dynamic Perspective. After having described the SAaaS4Mobile building block architecture, we can go into further details about the interaction among them from a dynamic, behavioural perspective. We identify three main phases of an end user-SAaaS4Mobile system interaction, as depicted in cf. Fig. 5.1:

i. *Sensors & Tasks Acquisition*: providing users with all available tasks, as offered by the SAaaS provider;
ii. *Sensor Use / Interaction*: selecting and preparing a task to be then submitted to the SAaaS provider, while keeping the ability to manage the task during its execution;
iii. *Observation Access*: in case one or more observations were the expected output of the task (e.g. scheduling or confirmed reservation), providing users with methods to retrieve stored measurements.

Resource release is not described here as it can be considered a special case degenerating from the management of requests for cancellation of tasks.

5.1. Sensors & Tasks Acquisition. The first macro-step of this workflow consists in the acquisition (enumeration) of all the tasks available over the full set of (on-board) sensing resources. More in detail, as depicted in cf. Fig. 5.2, an activity diagram (AD) for resource acquisition, a client sends a request, featuring requirements and preferences on the kind of needed sensing resources and corresponding range of tasks, to be submitted to the SAaaS4Mobile framework server exposed by the provider of choice. At this stage the high-level request gets mapped to standards-compliant constraints that can be easily be verifiable against enumerated resources and associated task types; the mechanisms for this mapping are out of the scope of this paper, probably the object of future investigation efforts. From the perspective of the contributor, e.g. mote-side, the job of the

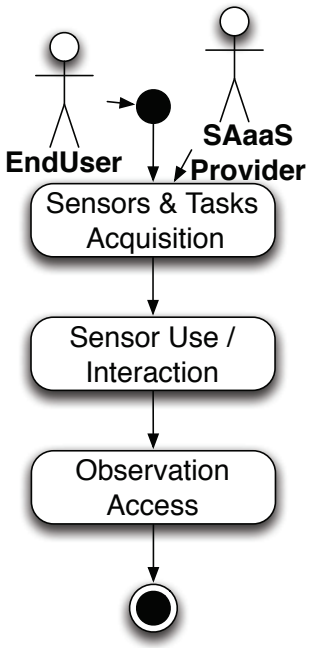


FIG. 5.1. *SAaaS4Mobile-end user interaction AD*

SAaaS4Mobile framework is to independently (i.e. at boot-up) probe the mote, at a level as close to hardware or OS / platform as possible, to find any exploitable sensing (or actuation) resource on board, e.g. not allocated exclusively to some other, immutable, activity. The probing thus happens at the very least in parallel, or possibly even long before the first request to be mapped comes in. The core resource (information) acquisition then happens by means of a two-step operation: the first being the search for capabilities, e.g. the kinds of phenomena the devices would sample during observations. The second one pursues the goal of retrieving all available tasks among the subset of sensing resources chosen by selection over advertised capabilities, i.e. according to one or more of the aforementioned criteria (e.g. phenomena to be sampled).

Again, upon reception of the list of available tasks, the SAaaS4Mobile framework server scans it to find a task matching the provided requirements. Moreover, the enumerated tasks provide a detailed description of parameters that can be set at the will of the customer. The list is sent as an endpoints' notification. If none correspondence was found, the SAaaS4Mobile framework server sends a notification to the client indicating that there is no results.

5.2. Sensor Use / Interaction. The second macro-step enables users to manage and configure a task, as obtained according to the first one. Therefore, the former can be split in its constituent macro-actions, and depicted accordingly in two ADs, the Submission (cf. Fig. 5.3) and Management (cf. Fig. 5.4) ones, respectively.

Submission operations, per the AD, comprise pre-submission configuration stages for a task, and submission itself. In the previous step of the high-level user-system interactions, the client has received a subset of available tasks, filtered by compatibility to constraints on capabilities and other requirements. At this point the client just has to choose one among the available alternatives for tasks, ready for reservation and submission, and finally configure it. At last, once the configuration is over, there are three different methods to submit the task, including configuration parameters, to the sensor:

- *direct submission* - a "Submit" request by the client gets forwarded to the sensor, being managed by the Task Submitter, while containing all the parameters needed to enable the resource to service the task under consideration.
- *submission by reservation* - reservation of a task for a beginning of processing stages in the future, under the guise of a "Reserve" request, which aims to book a resource (e.g. task) for a limited period

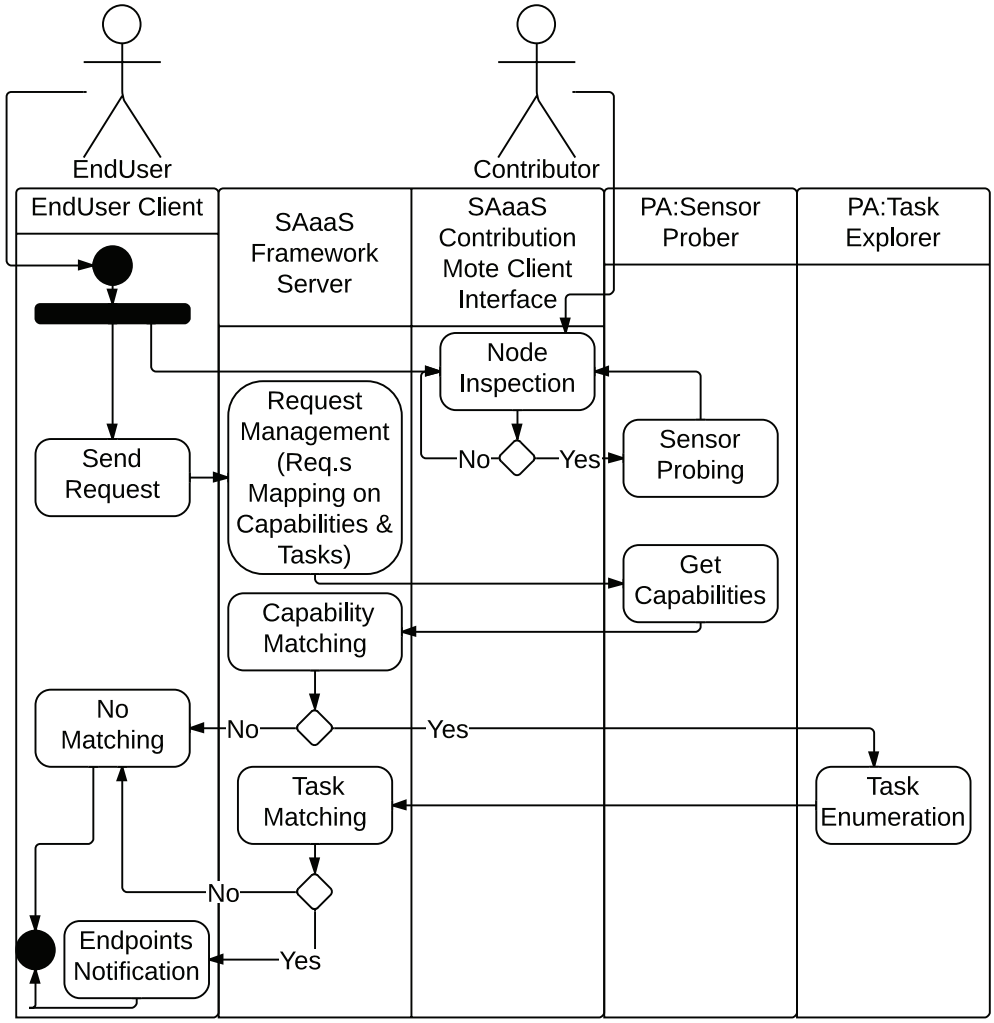


FIG. 5.2. *Resource acquisition AD*

under exclusive access. In case the resources are already allocated to another client or the configuration contains an error, no further progress can be achieved along the reservation attempt, apart from starting over. Otherwise, if the reservation was successful, a request to "Confirm" it may be sent, at the discretion of the client previously forwarding the reservation, upon which task processing commences, up to completion and subsequent deallocation of reserved resources, for the next request to be serviced. Both requests, i.e. reservation itself, and its confirmation, are managed by the Reservation Manager.

- *feasibility checking* - a "Get Feasibility" request, to be fully managed by the Feasibility Controller. The corresponding response signals approval or denial of subsequent submission / reservation operations, as evaluated at request time, thus dependent on availability of resources per conveyed requirements. If execution is evaluated as feasible, then the client can send a submit (or a reserve) request, and follow along one of the two aforementioned flows.

The AD related to Management describes the flows where the clients act upon an already running instance of a task execution, in particular empowered by three kinds of requests available at that stage:

- *status checking* -"Get Status" invocation to know at which step of the execution is the task;

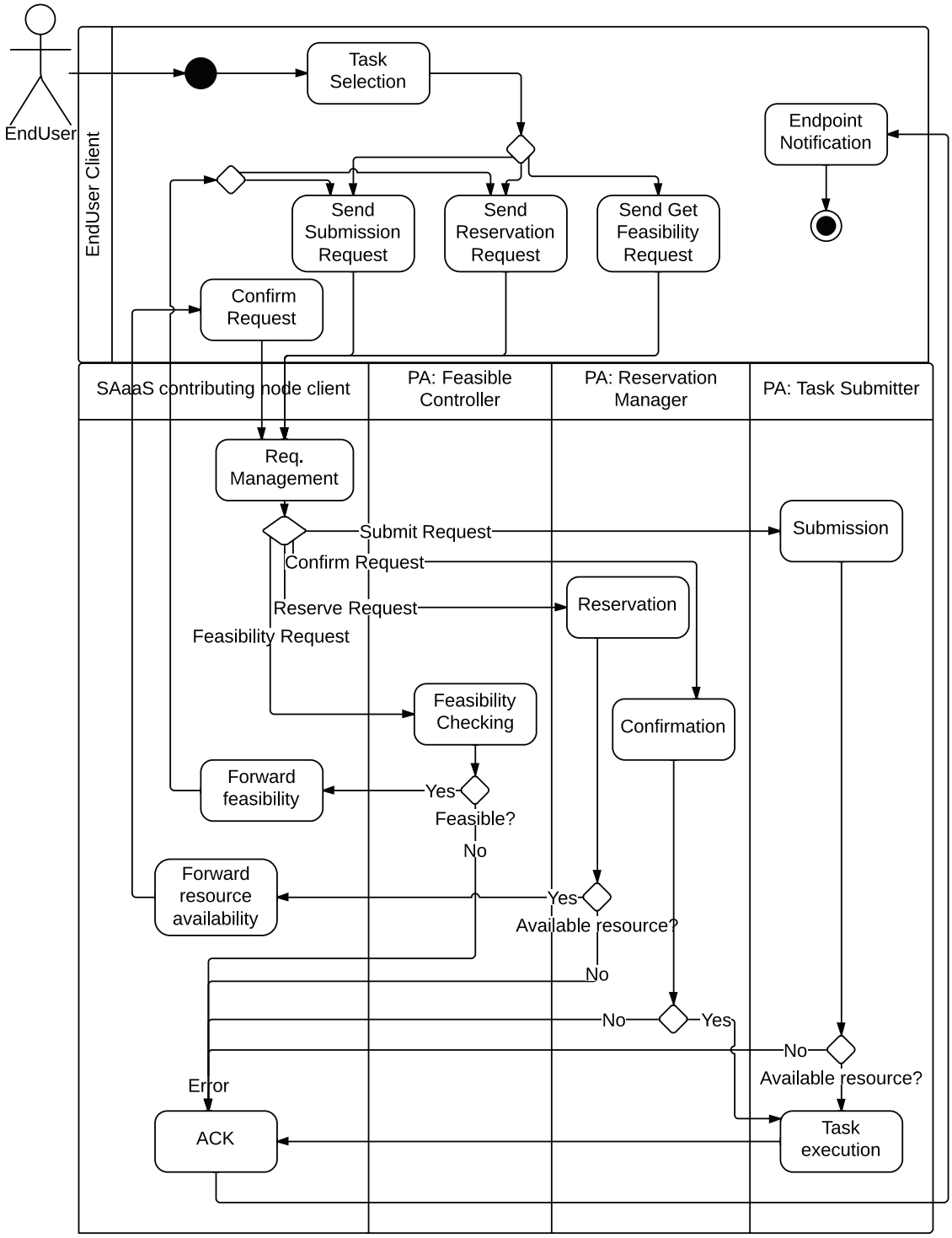


FIG. 5.3. *Interaction for submission operations*

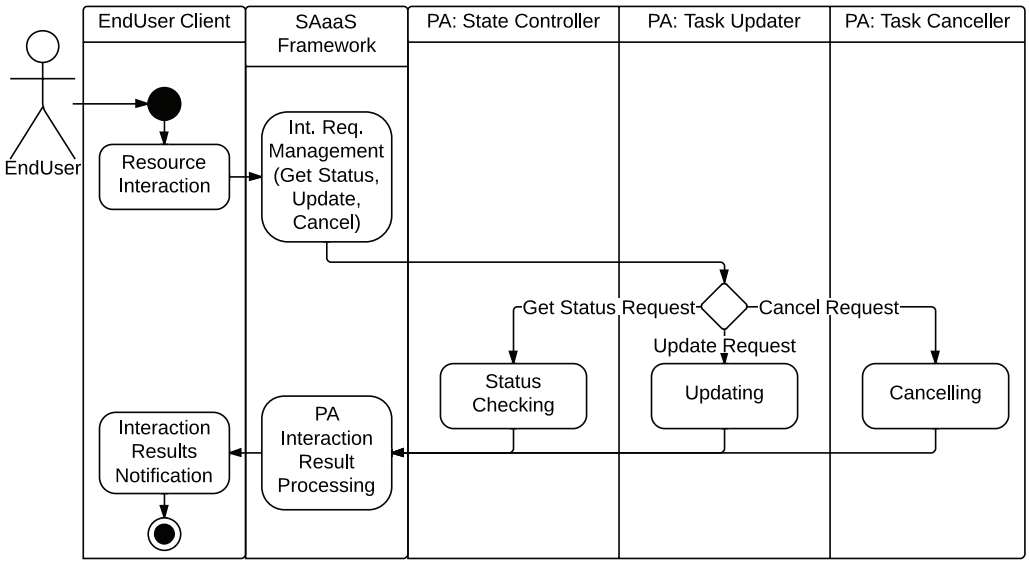


FIG. 5.4. *Interaction for management operations*

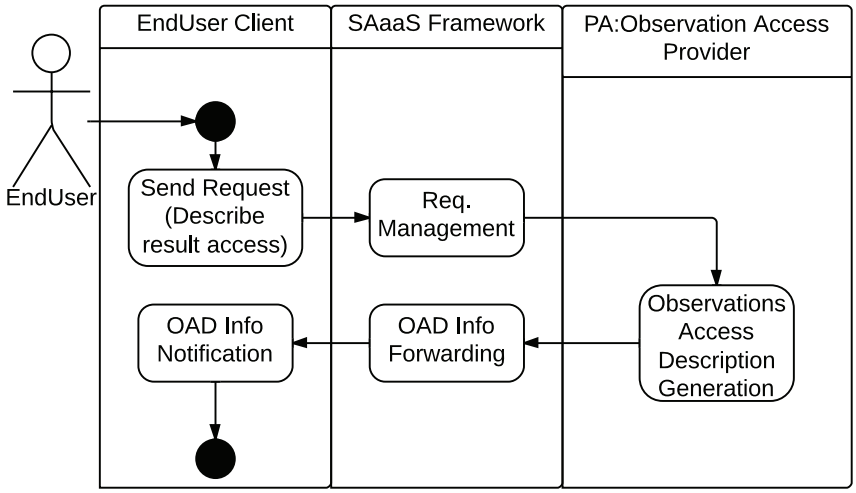


FIG. 5.5. *Observation access AD*

- *updating* - "Update" invocation to reconfigure task parameters;
- *cancelling* - a "Cancel" request to quit submitted or reserved tasks.

As can be seen by inspecting the swimlanes in the diagram of cf. Fig. 5.4, these requests are services respectively by: State Controller, Task Updater and Task Canceller, all components inside the Task Manager.

5.3. Observation Access. The last diagram, in cf. Fig. 5.5, depicts the flow for the user to get access to past observations, obtained thanks to the corresponding task(s). Indeed, as specified in cf. Sect. 4.2, the PA may also leverage Observation Agent services. So, after an interaction, when involving tasks to schedule observations, a client may later demand for endpoints and/or mechanisms to access data about obtained measurements, by means of "Describe Result Access" requests specifically. This translates to a transparent (to the user) interaction between the PA and the Observation Agent.

6. Proof of Concepts. In this section we detail on the implementation of a preliminary version of the SAaaS4Mobile framework including the very basic core abstraction modules and functionalities, which has been first described and then evaluated from an operational point of view.

6.1. Implementation. The implementation of the low-level modules of the SAaaS4Mobile framework has been targeted to mobiles equipped by Android OS 4.0, using the NDK developer libraries and API provided by the Android community [15]. The core of this effort, the design and coding of the Abstraction Unit, is based on the SWE Sensor Planning Service (SPS) 2.0 standard [22]. It enables the interaction among user clients and sensor and actuator services using XML schemas to submit requests and to allow the service to reply. Modeling behaviour after the SPS standard, the functionalities of the Sensor Prober, Task Explorer, Task Manager and Observation Access Provider modules described in cf. Sect. 4.2 have been developed.

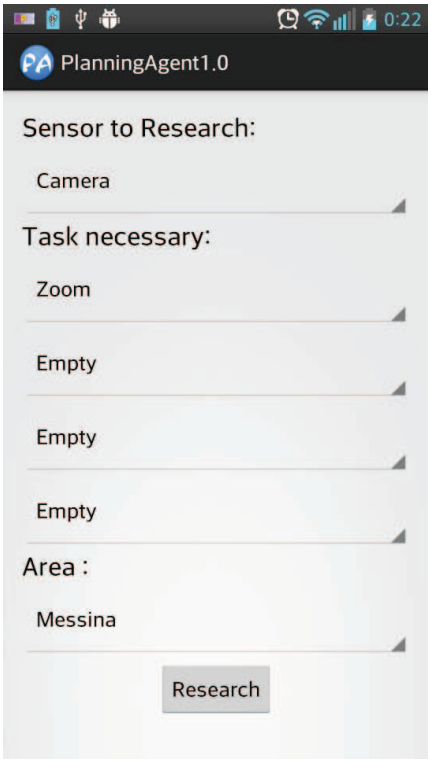
The Sensor Prober has to retrieve information regarding: i) the contributor, if available (the extent of such information disclosure is totally up to the contributor); ii) the node sensors and their descriptions, also including the measured phenomenon and corresponding metrics; and iii) the geographic area (range) inside which observations are significant. This feature is implemented by the SPS *GetCapabilities* primitive. A *GetCapabilities* request is composed of four sections. The first one is *ServiceIdentification* containing the contributing node metadata, i.e. generic info on the type of the node, brand, model and similar. Then the *ServiceProvider* section provides information on the contributor, if available and public. The third section is the *OperationsMetadata* one, with metadata about the operations specified by the service and implemented by the node. The last is the *Content* section, containing metadata about the sensors provided by the smart device through the PA and the communication mechanisms supported (XML, SOAP, etc.).

The Task Explorer retrieves the list of tasks that can be performed on a sensor through specific SPS *DescribeTasking* requests. A description of the available configuration operations for the sensor is thus obtained and provided to the Task Manager. As shown in cf. Fig. 6.1(a), the request just contains a *Procedure* element to enquiry a sensor in the list about the tasks that can be performed. The tasks are identified by the name, the description, and the capabilities' configuration information. The Task Manager implements a set of SPS requests. The *Submit* one allows the user to launch the execution of a configured task. Eventually, before to submit a job request, it is possible to enquire about its feasibility through the *GetFeasibility* primitive as shown in cf. Fig. 6.1(b). The reply, as depicted in cf. Fig. 6.1(c), can be "Feasible" or "Not Feasible" and, optionally, it may contain a list of alternative sets of tasking parameters that might help to the reformulation of a request. The user can also reserve the resources required to perform a specific task and then launch the task through the *Reserve* and *Confirm* requests as shown in cf. Fig. 6.2(a). In a *Reserve* request an expiration time has to be specified. At expiration time, all the reserved resource are released if the task has not been confirmed as in cf. Fig. 6.2(b).

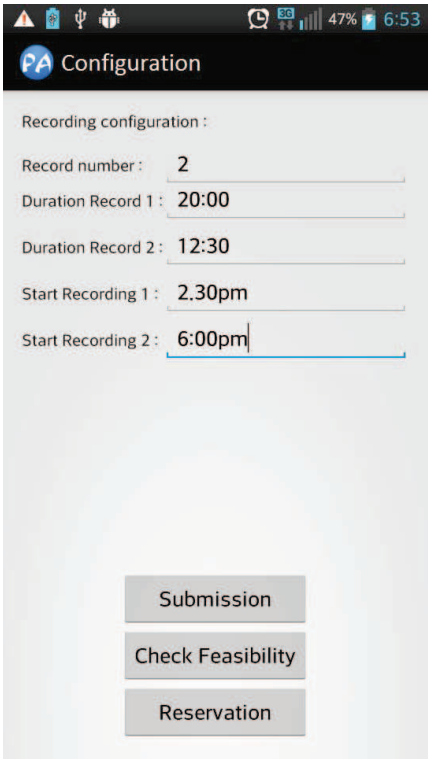
It is possible to check the status of a task using the *Status* request as shown in cf. Fig. 6.3(a). A task can be in six different states: "In Progress" if the service is executing it (cf. Fig. 6.3(b)), "Completed" if it was completed as planned, "Reserved" if it has been reserved, "Failed" if execution fails, "Expired" when the task reservation expires and "Cancelled" if the task was cancelled. The client can eventually update or cancel a task, with the *Update* and *Cancel* requests respectively.

Finally, the Observation Access Provider in the PA aims at providing the client with mechanisms, if needed, and endpoints to access the observations and measurements obtained during execution. It implements processing of SPS *DescribeTaskingResult* requests to interact with a specific sensor or a specific task as the ones shown in cf. Fig. 6.4.

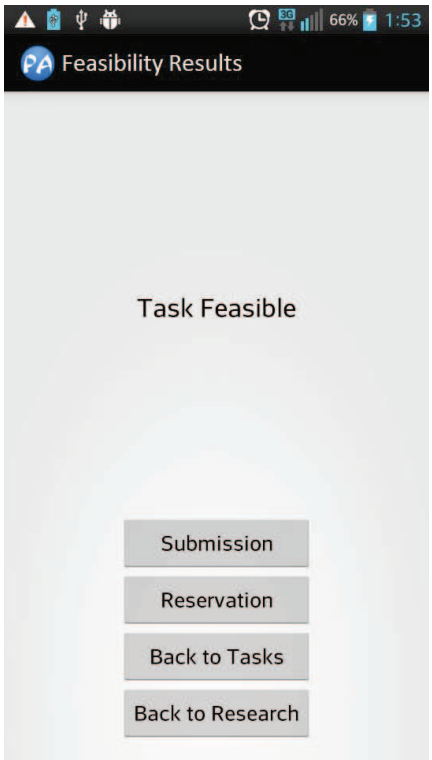
6.2. Preliminary Evaluation. In this section we provide some results on a preliminary prototype implementation of the SAaaS4Mobile abstraction layers. We have therefore implemented a mock SAaaS4Mobile testbed composed of an Intel I7 laptop acting as the SAaaS4Mobile server and an Android 4.2 Samsung S3 mobile as client. The SAaaS4Mobile client and server are implemented leveraging Java servlet technology using Apache Tomcat as the servlet container. The prototype is used to test the deployment and operation of very simple and basic operations as the ones discussed above. More specifically we evaluated *GetCapabilities*, *DescribeTasking*, *Submit* and *Observation Access* requests, by invoking them and iterating each test 1000 times, collecting the corresponding results to obtain the mean time and the standard deviation for each measurement as described in the following.



(a) Acquisition

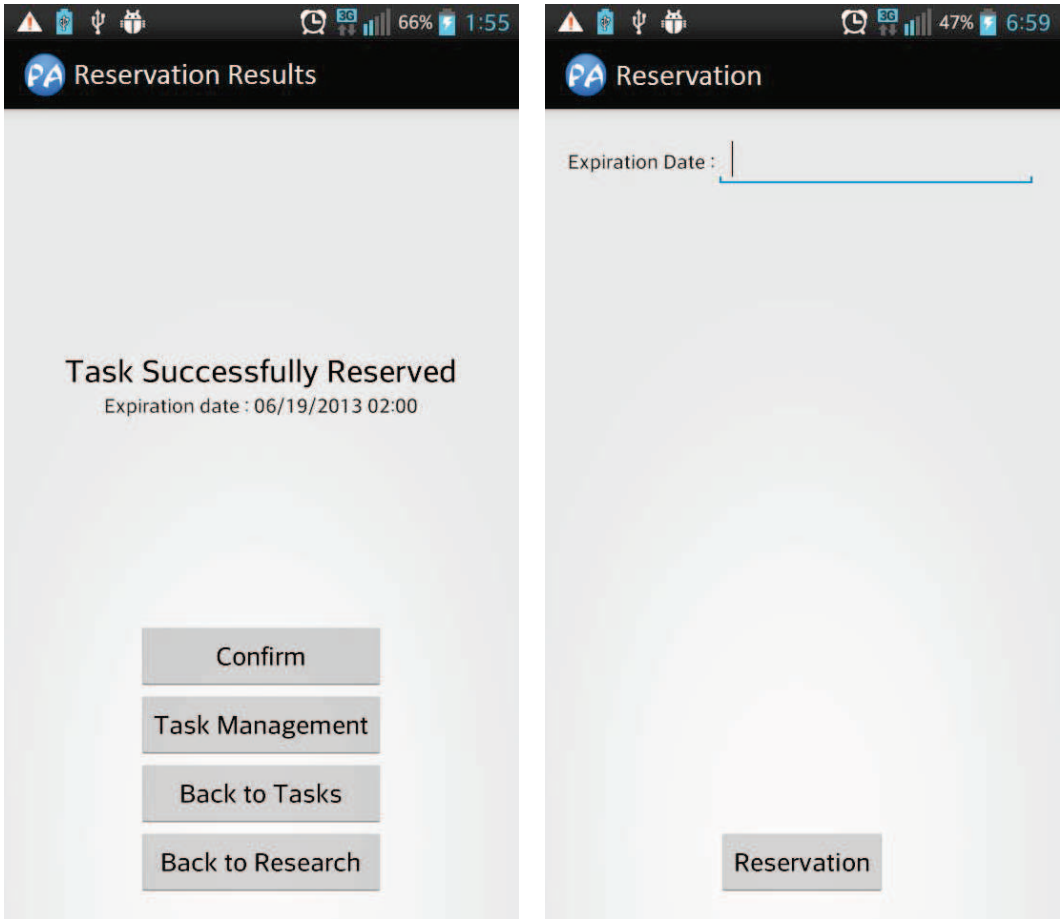


(b) Configuration



(c) Feasibility

FIG. 6.1. *SAaaS4Mobile resource acquisition, configuration and feasibility check*



(a) Reservation (b) Expiration timeout

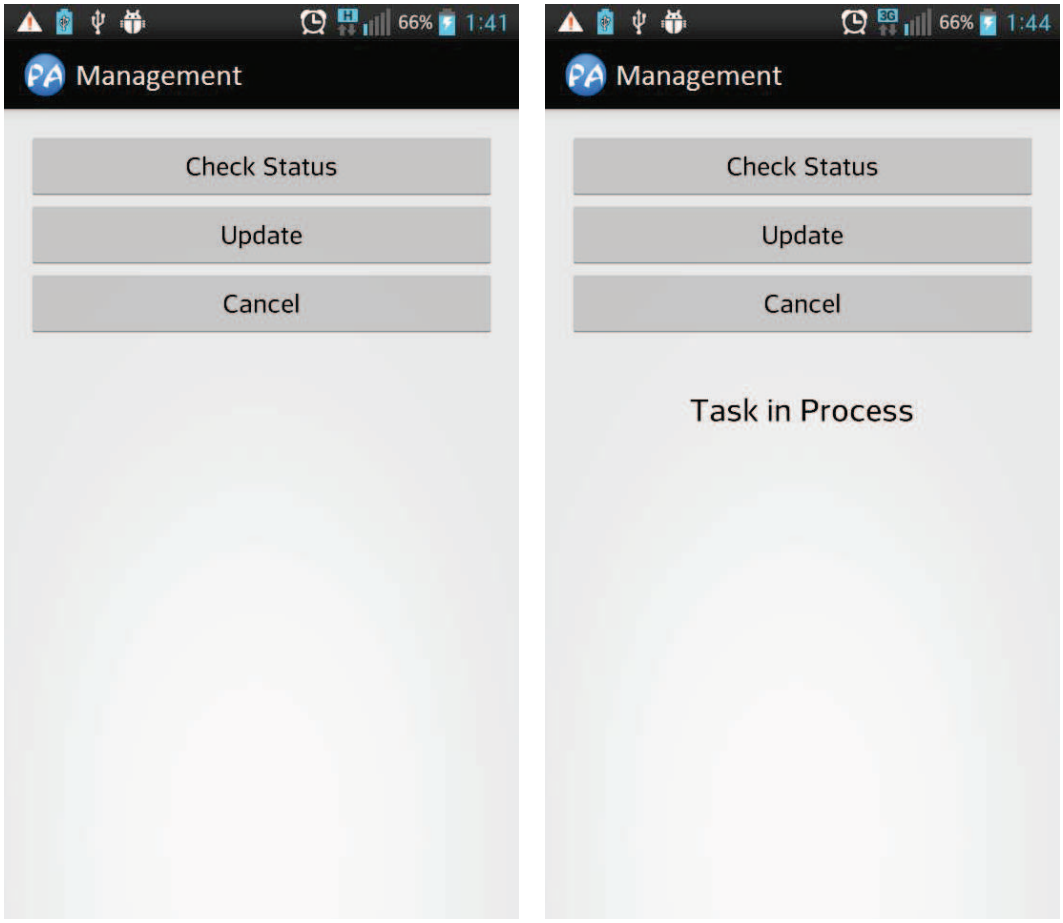
FIG. 6.2. *SAaaS4Mobile resource reservation and expiration timeout setting*

, and ***Parameter/ Statistic***	**GetCapabilities**	***DescribeTasking***	***Submit***	***Observation Access***
	ms	ms	ms	ms
μ	383.3	381.52	586.53	345.66
σ	11.2	12.1	20.5	9.7

TABLE 6.1
Basic operations response time obtained through the experiments.

All the parameters thus obtained through the evaluation are reported in cf. Table 6.1 . From these values we can argue that the most time-consuming operation is the *Submit* one, while the *Observation Access* request is that with the lowest delay. It could be also observed that both *GetCapabilities* and *DescribeTasking* have more or less similar performance. Thus, a whole workflow as the one depicted in Figure 5.1, made up of a sequence of invocations for the four aforementioned operations, has at least a response time of about 1700 ms.

These preliminary values strongly encourage us in furthering the development of the SAaaS4Mobile framework, since they serve as a foundation for assessing the feasibility of the SAaaS approach. A further and more comprehensive use case development, based on this preliminary implementation, is ongoing.

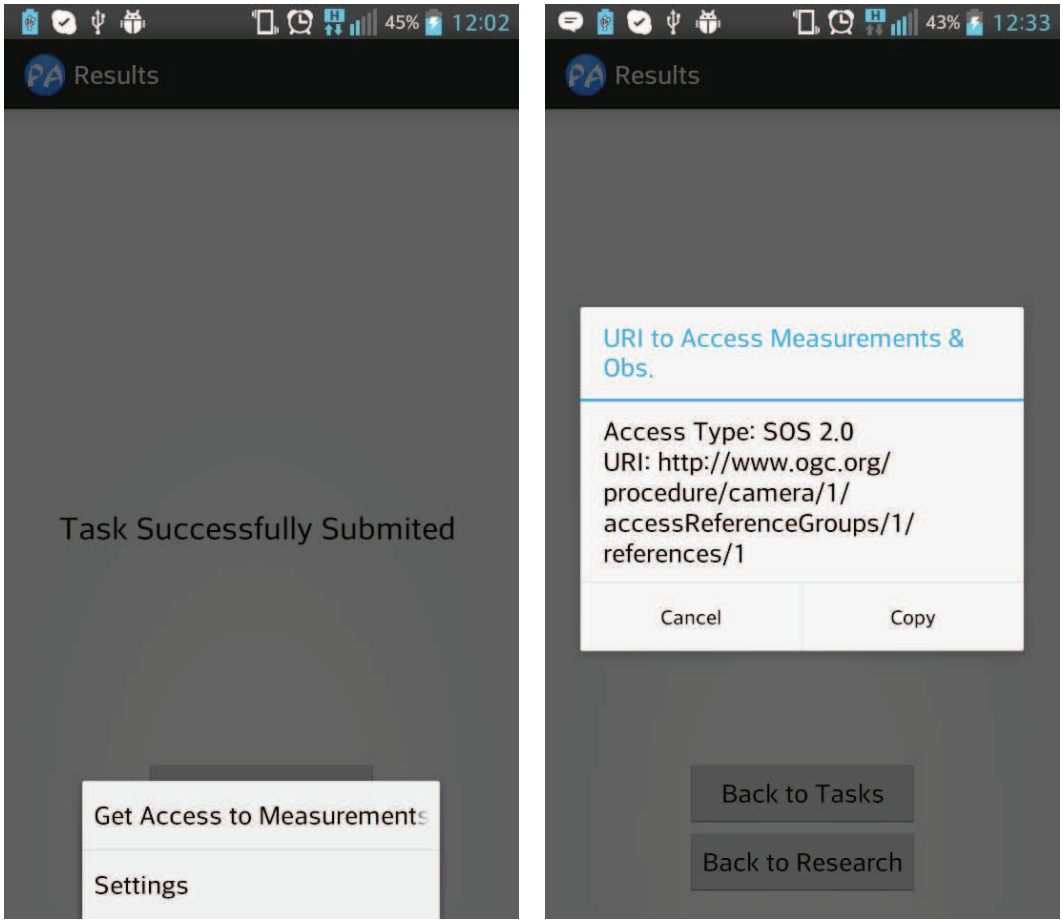


(a) Reservation (b) Expiration timeout

FIG. 6.3. *SAaaS4Mobile resource reservation and expiration timeout setting*

7. Conclusions. Aim of this paper is to continue presenting and discussing the feasibility of a sensing Cloud, able to actively involve devices, personal or standalone as well as grouped into specific administration domains such as sensor networks, either mobile or static. According to this vision, sensing and actuation resources, shared by device owners and administrators in a volunteer contribution fashion, are gathered by sensing Cloud providers to be provided on-demand, elastically, according to end-user requirements. This approach has been formalised into the Sensing and Actuation as a Service paradigm that, similarly to IaaS for compute Clouds, aims at providing actual, even if virtual, (sensing) resources. With regards to mobiles, this perspective complements and extends the one relative to mobile Clouds, where mobiles are just clients of Cloud-powered services, by actively involving them into a wide sensing infrastructure accessed and provided as a service. This way, the SAaaS paradigm lays at the intersection between the IoT and the service oriented/utility/Cloud computing fields.

To implement such a sensing Cloud several functionalities such as abstracting, virtualising, enrolling, collecting, discovering and managing (sensing and actuation) resources are required. Abstraction is a very basic one, since all nodes should be able to join the sensing Cloud and to communicate/interoperate, thus they should provide a uniform, abstract interface to underlying physical resources. In this paper we mainly focus on mobiles, dealing with issues related to sensing resource access and management through the SAaaS4Mobile



(a) Access request (b) Results URI

FIG. 6.4. *SAaaS4Mobile oservation access*

framework. In particular, the design of the SAaaS4Mobile Hypervisor module is tailored on mobiles' unique features, mainly specifying its blocks such as the Planning Agent, together with the communication layer inside the Node Manager and with the low-level access to devices the Translation Engine abstracts away.

Static and dynamic behaviour of these components have been described, both detailing their architecture and focusing on their interactions as well as on those between the end-user and the contributing node, i.e. the SAaaS4Mobile Hypervisor client. We also described these interactions and commands, from the angle of our prototype implementation on Android smartphones, also testing some on them through a proof of concepts demonstrating the feasibility of the approach.

Further endeavours are going to investigate and explore aspects related to virtualization, as well as to port SAaaS implementations to SNs. We are also eager to spend efforts over use cases and application scenarios, especially to carry out useful evaluations on performance, trying to validate the SAaaS approach by uncovering outstanding advantages and exploring unique features, by extending the current SAaaS4Mobile implementation. Moreover, we are also investigating on further developments of the approach in context of IoT, considering it as the implementation of a utility vision for the IoT paradigm, able to support novel, up and coming trends such as crowdsensing.

REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi, *Infrastructure for data processing in large-scale interconnected sensor networks*, in Mobile Data Management, 2007 International Conference on, may 2007, pp. 198 –205.

[2] M. Avvenuti, P. Corsini, P. Masci, and A. Vecchio, *An application adaptation layer for wireless sensor networks*, Pervasive Mob. Comput., 3 (2007), pp. 413–438.

[3] M. Behan and O. Krejcar, *Modern smart device-based concept of sensoric networks*, EURASIP Journal on Wireless Communications and Networking, 2013 (2013), p. 155.

[4] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa, *Evaluating wireless sensor node longevity through markovian techniques*, Computer Networks, 56 (2012), pp. 521–532.

[5] Z. Chen, N. Chen, L. Di, and J. Gong, *A flexible data and sensor planning service for virtual sensors based on web service*, Sensors Journal, IEEE, 11 (2011), pp. 1429–1439.

[6] J. Clarke, J. Lethbridge, R. Liu, and A. Terhorst, *Integrating mobile telephone based sensor networks into the sensor web*, in Sensors, 2009 IEEE, 2009, pp. 1010–1014.

[7] V. D. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa, *Cloud@home: Bridging the gap between volunteer and cloud computing*, in Emerging Intelligent Computing Technology and Applications, D.-S. Huang, K.-H. Jo, H.-H. Lee, H.-J. Kang, and V. Bevilacqua, eds., vol. 5754 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 423–432.

[8] S. Distefano, *Evaluating reliability of wsn with sleep/wake-up interfering nodes*, Int. J. Systems Science, 44 (2013), pp. 1793–1806.

[9] S. Distefano, G. Merlino, and A. Puliafito, *Sensing and actuation as a service: A new development for clouds*, in Proceedings of the 2012 IEEE 11th International Symposium on Network Computing and Applications, NCA '12, Washington, DC, USA, 2012, IEEE Computer Society, pp. 272–275.

[10] S. Distefano, G. Merlino, A. Puliafito, and A. Vecchio, *A hypervisor for infrastructure-enabled sensing clouds*, in IEEE International Conference on Communications, Budapest, Hungary, June 9-13, 2013 2013.

[11] S. Distefano and K. S. Trivedi, *Non-markovian state-space models in dependability evaluation*, Quality and Reliability Eng. Int., 29 (2013), pp. 225–239.

[12] M. Fazio, M. Villari, and A. Puliafito, *Sensing technologies for homeland security in cloud environments*, in Sensing Technology (ICST), 2011 Fifth International Conference on, 28 2011-dec. 1 2011, pp. 165 –170.

[13] M. Gaynor, S. L. Moulton, M. Welsh, E. LaCombe, A. Rowan, and J. Wynne, *Integrating wireless sensor networks with the grid*, IEEE Internet Computing, 8 (2004), pp. 32–39.

[14] G. Gil, A. Berlanga de Jesus, and J. Molina Lopez, *incontexto: A fusion architecture to obtain mobile context*, in Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on, 2011, pp. 1–8.

[15] Google Inc., *Android ndk - http://developer.android.com/tools/sdk/ndk/index.html*.

[16] ———, *Android Website - http://www.android.com/*, 2013.

[17] V. Huang and M. Javed, *Semantic sensor information description and processing*, in Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on, 2008, pp. 456–461.

[18] J. Jamsa, M. Luimula, J. Schulte, C. Stasch, S. Jirka, and J. Schoning, *A mobile data collection framework for the sensor web*, in Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS), 2010, 2010, pp. 1–8.

[19] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, *Virtual sensor networks - a resource efficient approach for concurrent applications*, in Information Technology, 2007. ITNG '07. Fourth International Conference on, april 2007, pp. 111 –115.

[20] Y. Liu, D. Hill, A. Rodriguez, L. Marini, R. Kooper, J. Myers, X. Wu, and B. Minsker, *A new framework for on-demand virtualization, repurposing and fusion of heterogeneous sensors*, in Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems, CTS '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 54–63.

[21] M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, *Vitro architecture: Bringing virtualization to wsn world*, in Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, MASS '11, Washington, DC, USA, 2011, IEEE Computer Society, pp. 831–836.

[22] Open Geospatial Consortium, *OGC(R) Sensor Planning Service Implementation Standard*, OGC, 2.0 ed., 2011.

[23] C. Reed, M. Botts, J. Davidson, and G. Percivall, *Ogc(r) sensor web enablement: overview and high level achhitecture.*, in Autotestcon, 2007 IEEE, sept. 2007, pp. 372 –380.

[24] L. Sarakis, T. Zahariadis, H.-C. Leligou, and M. Dohler, *A framework for service provisioning in virtual sensor networks*, EURASIP Journal on Wireless Communications and Networking, 2012 (2012), p. 135.

[25] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, *Hourglass: An infrastructure for connecting sensor networks and applications*, tech. report, 2004.

[26] L. Sun, D. Zhang, and N. Li, *Physical activity monitoring with mobile phones*, in Toward Useful Services for Elderly and People with Disabilities, B. Abdulrazak, S. Giroux, B. Bouchard, H. Pigot, and M. Mokhtari, eds., vol. 6719 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 104–111.

[27] M. Yuriyama and T. Kushida, *Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing*, in Network-Based Information Systems (NBiS), 2010 13th International Conference on, sept. 2010, pp. 1 –8.

[28] Y. Zhi-An and M. Chun-Miao, *The development and application of sensor based on android*, in Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on, vol. 1, 2012, pp. 231–234.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014